Preprint typeset in JINST style - HYPER VERSION

A Development Environment for Visual Physics Analysis

H.-P. Bretz, M. Brodski, M. Erdmann, R. Fischer, A. Hinzmann, T. Klimkovich, D. Klingebiel, M. Komm, J. Lingemann, G. Müller, T. Münzer, M. Rieger, J. Steggemann, T. Winchen

RWTH Aachen University, Physikalisches Institut 3A, 52062 Aachen, Germany
E-mail: erdmann@physik.rwth-aachen.de

ABSTRACT: The Visual Physics Analysis (VISPA) project integrates different aspects of physics analyses into a graphical development environment. It addresses the typical development cycle of (re-)designing, executing and verifying an analysis. The project provides an extendable plug-in mechanism and includes plug-ins for designing the analysis flow, for running the analysis on batch systems, and for browsing the data content. The corresponding plug-ins are based on an object-oriented toolkit for modular data analysis. We introduce the main concepts of the project, describe the technical realization and demonstrate the functionality in example applications.

KEYWORDS: Analysis and statistical methods; Software architectures (event data models, frameworks and databases); Data processing methods.

Contents

1. Introduction

Physics data analysis aims at extracting information from data taken by an experimental apparatus. The analysis and understanding of the data is typically an interplay of designing (and implementing) algorithms and of interpretation of the results. As the amount of data and the complexity of analyses tend to increase in modern experiments, the necessary amount of time for the technical integration of the individual analysis steps rises as well. The intention of Visual Physics Analysis (VISPA) is to increase the available time for the interpretation of the results by reducing the time spent on the technical aspects of the data analysis. The VISPA project provides a graphical development environment for physics analyses, including a selection of plug-ins specifically designed for the needs of data analyses in high energy physics (HEP) and astroparticle physics. The target group ranges from undergraduate students to experienced scientists. It has been continuously developed since 2006 [1–9].

In the field of high energy physics, a graphical analysis development environment constitutes a new approach. Most high energy physics collaborations provide dedicated software frameworks for different tasks ranging from reconstruction to user analysis [10–12]. While these frameworks attempt to provide solutions for multiple aspects of data processing, the VISPA environment is designed as a standalone application dedicated to high-level analysis with frequent development cycles using a reduced data format. Visual support is provided for the development, execution and verification of analysis workflows. Specific tasks like plotting, histogramming, and statistical analysis are not at the core of this development environment; they are, e.g., accessible via the `ROOT` framework [13].

We define a list of requirements for a development environment for physics analyses, which can be categorized into three groups. The first group of requirements is due to the complexity of modern physics analyses. A proper management of the analyses needs to be ensured by providing a structural basis, objects to represent an analysis, and development tools to keep track of the analysis structure. At the same time, the physicists should not be limited to predefined analysis techniques, conserving the freedom of scientists to create and develop new ideas. Therefore, useful functionality should be provided by the development environment, without invoking restrictions on analysis techniques. Another important requirement is an acceptable amount of time for the iteration of the analysis cycle, which typically consists of an analysis design phase, program execution, data verification, and re-design of the analysis.

The second group of requirements is related to the wide range of programming skills of the analyzing physicists. A fast start in physics analyses helps analyzers with little programming knowledge. It is advantageous to learn through intuitive and self-explanatory design. Appropriate guidance by the provided structures and interfaces as well as good transparency of the analysis help to avoid errors from user code, which is important also for the experienced physicists.

The third group of requirements is driven by the fact that many physics analyses are carried out in teams. The analysis development environment needs to support and facilitate this teamwork. This implies that analyses need to be easily exchangeable between physicists and their individually preferred working environments and especially operating systems. Visual representation of an analysis is desired to ease communication between analysts. Code and algorithms should be reusable in other ongoing or future analyses.

This article continues with a description of the project structure and the design decisions to meet the requirements for a development environment for physics analyses. The subsequent section gives an overview of the implementation of the VISPA graphical user interface. Afterwards, the three main components needed to design physics analyses are explained, namely a collection of physics objects and algorithms, a framework supporting modular physics analyses, and the graphical tools for physics analysis. Finally, example applications and extensions are detailed, followed by the conclusions.

2. Analysis Development Environment

The general structure of the VISPA development environment is depicted in Figure 1. The central entry point to physics analysis is the VISPA graphical user interface (GUI). Tools for physics analysis development are provided as plug-ins. The three major plug-ins that allow for the complete handling of physics analyses are the *Analysis Designer*, the *Data Browser* and the *Batch Manager*. With the *Analysis Designer*, physics analyses can be designed and executed. The *Data Browser* is a tool to browse the data input and output written in the file format of the Physics eXtension Library (PXL) [14]. Both the *Analysis Designer* and the *Data Browser* are based on PXL, which is a part of the VISPA project. PXL is a C++ toolkit that provides physics objects, data input and output, a module system, and other tools to program physics analyses. The *Batch Manager* is able to configure batch jobs, in particular from analyses designed with the *Analysis Designer*, and to send them to a selected batch system. Further tools, e.g., browsers for experiment-specific workflows, can be added to the graphical development environment with the plug-in system.

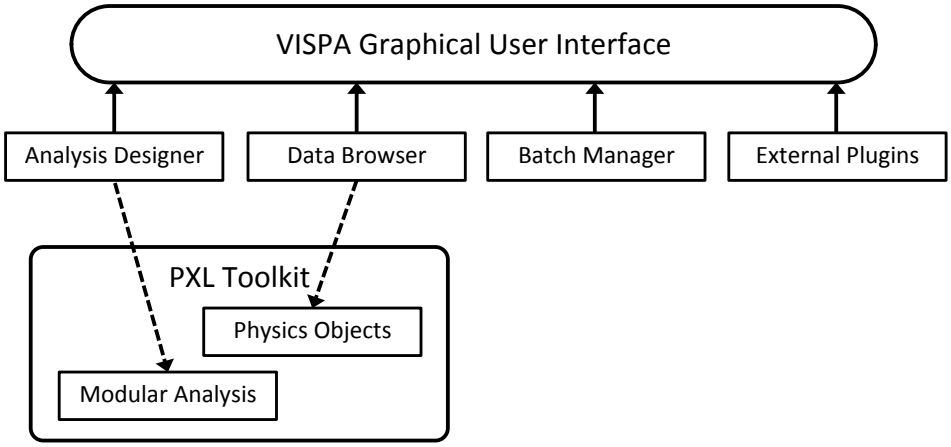


Figure 1. Structure of the VISPA development environment. The central entry point to physics analysis is the VISPA GUI. Tools for physics analysis development are provided as plug-ins. The analysis-specific plug-ins, namely the *Analysis Designer* and the *Data Browser*, are based on the functionalities of the PXL toolkit. External plug-ins from collaborations and users can be plugged in the VISPA environment as well.

To fulfill the requirements given in the introduction, several design choices build the basis for VISPA and are described in the following. The first choice is bundling of the iterative analysis development process, consisting of analysis prototyping, execution and verification of the results, into a single development environment. This integrated design gives rise to a good manageability of the full analysis cycle.

Second, instead of pre-defining analysis solutions by providing each single building block of an analysis, VISPA provides tools for the user to design an analysis structure. The algorithms are input by the user or a collaboration.

Third, VISPA is designed to be extendable. This is realized via the plug-in mechanism, allowing the inclusion of plug-ins that are based on the VISPA GUI. By enabling experiment-specific extensions of the VISPA development environment, it is possible to handle the complete analysis workflow including steps involving software from the experiment in a single development environment.

Fourth, VISPA is designed to run on all major operating systems, i.e. Linux, Microsoft Windows and Mac OS X. This feature is particularly important to enable the sharing of algorithms between users and the ability to transport entire analyses. The VISPA GUI is therefore based on the platform-independent GUI framework `PyQt4` [15, 16].

3. Graphical User Interface

The core components of the VISPA GUI are the classes *Application* and *MainWindow* as depicted in Figure 2. They provide common functionality for opening and closing files, creating corresponding menu entries, and opening and closing tabs. A single interface to this functionality for the plug-ins is provided by the class *GuiFacade*. On startup of the application, a *PluginManager* searches for

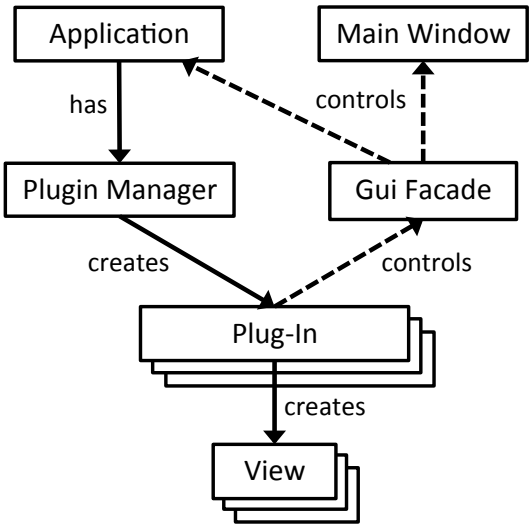


Figure 2. Structure of the VISPA GUI implementation. The full VISPA GUI class structure is documented in [17].

available plug-ins and creates the corresponding instances. The plug-ins themselves can create new tabs via a singleton object of the class *GuiFacade*, e.g., on a file open request from the Application.

For the graphical display of objects within tabs, such as analysis modules or data objects, a mechanism for data-model driven *Views* is introduced in VISPA. By defining a common interface between data objects and views, a single view can be reused for different data objects. In addition, the same data objects can be displayed in different views, e.g., to represent different aspects of the object. VISPA supplies a set of views covering a large variety of use cases in custom applications. Experiment-specific views can also be implemented based on the existing views.

While the VISPA GUI is solely dependent on `Python` and `PyQt`, custom views and plug-ins may dynamically use other external dependencies. An example for a view using an external library is the *RootCanvasView*, which is capable of displaying graphs produced with `ROOT`, e.g., the distribution of particle momenta in the angular plane.

For the development of new views, a variety of shared components is available. These include useful graphics components, such as connectable boxes and lines, a common zooming mechanism for all components in VISPA and the functionality of each compound to be exported to various raster or vector graphic formats (e.g. `PostScript`).

The VISPA GUI and its handling are designed with reoccurring patterns for good manageability and a fast understanding of the structure. An example for a reoccurring pattern is the three-column representation of analyses in the *Analysis Designer* and of data files in the *Data Browser* as depicted in Figure 3. In the *Analysis Designer*, the left column presents available modules for analysis, whereas in the *Data Browser*, the data objects in a file are displayed in a tree structure. The center column holds a graphical representation of the analysis or the data, respectively. Depending on the view, this could for instance be a Feynman-like representation. The properties of an object selected in the center column are displayed in a *Property View* in the right column.

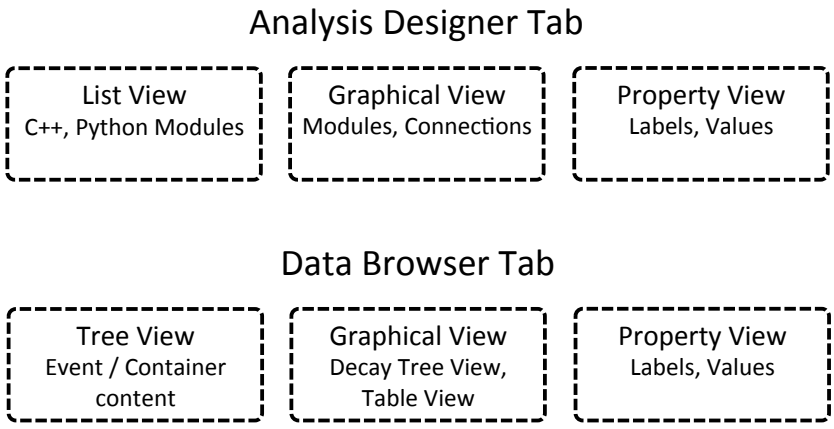


Figure 3. Stylized layout of the *Analysis Designer* and *Data Browser* tabs.

4. Physics Library

In the VISPA environment, analyses are constructed based on classes representing physics objects and algorithms provided by the C++-based toolkit PXL, the successor of the PAX toolkit [9,18–21]. PXL offers a variety of classes for physics objects and algorithms as well as tools for code and analysis handling. The individual components, described in the following, are designed to be used independently as tools for physics analysis or within the VISPA environment.

PXL contains classes to represent objects from high energy physics, from astroparticle physics and for general purposes. An example from HEP is the class *Particle* which contains a four-momentum and properties such as the particle charge. For the field of astroparticle physics, e.g., classes representing ultra-high energy cosmic rays (UHECR) are available together with common operations such as transformations between astrophysical coordinate systems. General-purpose classes include matrix and vector representations.

An excerpt of the class structure of selected HEP objects in PXL is shown in Figure 4. For HEP, two base classes are provided from which the physics objects and user-defined objects can inherit. The *Object* provides user-specific data and the *Relative* can have relations with other objects, which are explained later in this section. Similarly, base classes for astroparticle physics objects are provided. All objects carry universally unique identifiers (UUIDs [23]) which are used to identify C++ classes (type ID) as well as each individual object (object ID).

PXL provides different types of containers to group various PXL objects. The *BasicContainer* is a collection of PXL objects that takes ownership and deletion responsibility for the inserted objects. The *Event* provides an extended container specifically designed for holding HEP objects. An example are particles of a decay tree which have mother-daughter relations. Deleting the *Event* implies automated deletion of the particles contained in the event and their relations. Furthermore, PXL provides containers which themselves can have relations with other objects. A specific use case from HEP is an *Event* holding a set of containers representing different views of the same

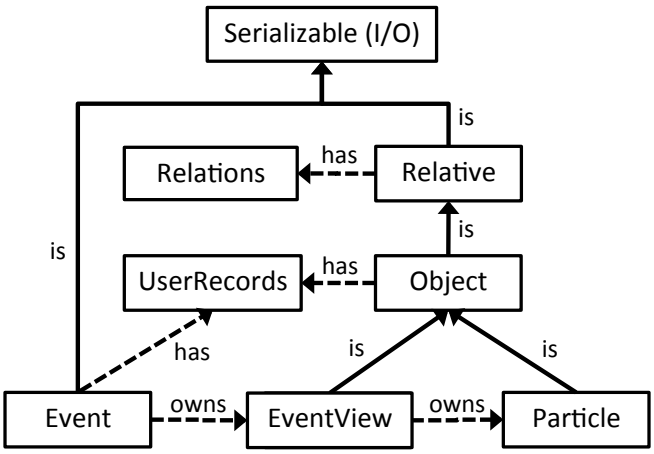


Figure 4. Excerpt of the class structure of physics objects in PXL showing the structure of selected HEP objects. The full PXL class structure is documented in [22].

physics event, so called *EventViews*, e.g., if there are ambiguities in the reconstruction leaving room for different interpretations.

A key feature in PXL to provide full flexibility in the implementation of analyses is the possibility to extend any object by user-specific data. Each *Object* has a *UserRecords* instance which is a map of string-indexed entries of generic type. Generic types are represented by a variant data type that covers basic C++ types, the STL string representation and any PXL object type.

Relations between physics objects can be realized in PXL through two different mechanisms depending on the use case. Firstly, relations of objects within the same container (e.g. particles in an event) are realized in such a way that the container takes care of the deletion of its relations between them to ensure full consistency at all times. A *Relative*, for example, has two *Relations* objects to implement mother and daughter relations. Secondly, so called soft relations are realized by a map of string-indexed relations to generic objects even outside a container through their unique object ID.

The PXL objects come with an input/output (I/O) scheme where each object is decomposed into serially written basic types. All objects inheriting from the class *Serializable* define how to (de-) serialize themselves, and can thus be used in the PXL I/O scheme. The I/O is managed in a set of classes that provide the user interface for writing and reading files, gathering data chunks, managing compression, controlling the buffer storage and managing the basic type I/O. PXL I/O specifically uses 32 or 64 bit variables to correctly support both systems. All data is written to disc in little endian format, as this is most commonly used. On big endian systems all data is converted. The PXL I/O aims for robustness and simplicity and is designed for simple splitting and merging of data at the file level.

To allow the use of C++ and Python code within the same analysis, the full C++ interface of all PXL classes is made available in Python. The Python interfaces are wrapped around the C++ classes automatically using `Swig` [24]. Automatic conversion between C++ and Python types is

also provided such that all PXL objects can be handled like native Python objects. The Python interface of PXL allows full introspection of all properties and methods of any object and also supplies descriptions for all of these. Full introspection of all objects in PXL enables the *Data Browser* plug-in of VISPA to provide visual inspection of all properties of any PXL object.

For easier code and analysis handling, a set of convenience mechanisms is provided by PXL. For example, a flexible and uniform logging mechanism is available for consistent command-line and log-file output from within the whole analysis.

Finally, a set of general-purpose algorithms is included in PXL: a node-based automatic layout, which is, e.g., used in VISPA to display particle decay trees; an automated reconstruction of all possible permutations of decay trees [25]; and the sorting as well as filtering of object collections.

For the longterm maintenance of the PXL and VISPA code, a server based on the `redmine` [26] management web application is set up, providing tools for bug tracking, project planning and code versioning using `mercurial` [27]. Code development for multiple platforms is supported; the correct functionality is ensured by the usage of continuous integration with automatic builds and the execution of unit tests to validate the code on all target platforms.

The components of PXL provide the infrastructure to build complex analyses in an object-oriented design. The objects for HEP and astroparticle physics in C++ and Python, including the concepts of user data, relations, containers and I/O scheme, are designed to fulfill the requirements of good performance and easy handling.

5. Modular Analysis System

The PXL module system provides the interfaces and runtime environment to create and run physics analyses with individual modules. It is the underlying analysis system of the *Analysis Designer* in VISPA, but it can also be used independently of the VISPA development environment. In the analysis, data are handed through a chain of modules. Each module can have multiple sinks for data input and sources for data output. The modules contain the algorithms to process the data and steer it to different sources. This implies that the execution of the modules is based on the flow of the data through the analysis chain.

An example of a module chain is depicted in Figure 5. The ‘Input A’ and ‘Input B’ modules trigger the execution by reading a data chunk and passing it to the next module, an ‘Analyse’ module that may, e.g., run an algorithm on the data and add information to it. In the next module, labeled ‘Decide’, analysis logic can be implemented by either sending the data to the ‘yes’ or the ‘no’ source and thereby deciding to which output module the data go. In addition to the possibilities demonstrated in the figure, each sink can receive input from multiple sources. The processing of the current data object stops if a module does not pass the object to any of its sources or if the source is not connected to another module. By default, all modules able to start an analysis chain, e.g., input or generator modules, are run in a single loop over the data. Alternatively, by assigning a run index, they can be grouped into several loops over the data executed sequentially. The input modules are able to read a list of files. The use of more than one input module is in particular useful if categories of data are to be treated differently in the module chain.

For an appropriate overall turn-around time of the analysis development, it is important to be able to optimize computing-intensive tasks for high code performance and less intensive tasks for

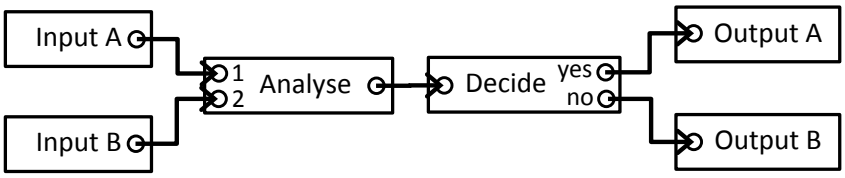


Figure 5. Exemplary module chain. Data are read in by the 'Input A' and 'Input B' modules and are sent to the next module 'Analyse'. A 'Decide' module splits the data stream, which is finally written into two different output files.

the minimum time spent by the user for code development. One crucial design choice is therefore the use of two programming languages in the module system. Users can freely choose to program their modules in C++ or Python and combine them in the analysis chain. C++ is the de facto standard in HEP and astroparticle physics [10–13]. Python is a widely used scripting language in science [13, 28], which is often used to glue together code from different programming languages like C and Fortran [24]. While C++ code can be highly optimized for performance, Python as a scripting language allows fast and simple implementation of analysis logic. A fast start into analysis using Python is well suited for beginners, while fast code prototyping in Python as well as high performance C++ coding can be achieved by experienced users.

By the use of C++ and Python, the module system is designed to be interfaced to a large variety of analysis- and experiment-related software. External software is accessible through their C++ or Python interfaces from any analysis module. This includes plotting packages (e.g. `ROOT` [13], `matplotlib` [29]), statistical analysis tools (e.g. `RooStats` [30], `RooFit` [31]) and math/algebra packages (e.g. `SciPy` [28], `NumPy` [32]). Furthermore, interfaces or converters for data input from various experiments (CMS, ATLAS, Pierre Auger, ILC, ...) and file formats (LHE [33], CSV [34]) have been implemented.

The data passed through modules along the chain can be any derivative of a PXL class. For physics analyses in HEP, typically, a container object representing a particle collision event holding a set of observed particles is used. For astroparticle physics analyses a container object carrying a dataset of ultra-high energy particles is used.

All analysis modules implement an abstract interface provided in the module system. It defines the access methods to data sinks, data sources, and module options; it also defines the methods executed on each data package before, during and after the execution of the analysis. The module interface is identical in C++ and Python, thus simplifying the conversion of fast-prototyped Python modules into high-performance C++ modules. VISPA is delivered with a set of predefined modules for file input/output and module skeletons for analysis logic (generate, decide, switch and analyse), which are typically combined to an analysis with user-defined modules.

The analysis structure, including the modules as well as their connections, is stored in an object of the class *Analysis*. This class provides the functionality to store/retrieve an analysis setup to/from an XML file. The class *Analysis* can be used by two different executables: *pxlrun*, which performs command line execution of analyses, and the graphical environment of VISPA, which

allows the visual design and execution of analyses. The full analyses including modules and data can be exported into an archive file from the VISPA environment, which can be used to exchange analyses with others and for batch processing.

The definition of a common interface for the exchanged data (e.g. Event for HEP) ensures the exchangeability of data between users. The common interfaces of the modules and the *Analysis* ensure the exchangeability and reusability of modules and analyses. In particular, common modules can be shared within analysis groups, thereby facilitating teamwork.

The flow-based design is particularly well-suited for HEP analyses where particle collision event data are processed. By analyzing the particles and other information in an event, decisions are typically taken whether to select certain events as depicted in Figure 5.

A specific example for astroparticle physics analyses in VISPA are studies of extragalactic magnetic fields through ultra-high energy cosmic rays with the Pierre Auger Observatory [35]. Due to the experiment-independent format, VISPA can be used to transfer analyses from experiment to experiment. The analyses of Pierre Auger data constraining extragalatic magnetic fields can in principle be directly applied to a dataset from other cosmic ray experiments.

6. Using the Components of the Development Cycle

The process of analysis development and application in VISPA is centered around a graphical representation of the analysis in the *Analysis Designer*. This supports the analyzer in designing a well-structured and modular analysis and at the same time helps to structure his analysis work. The main analysis tasks, like browsing input or output data in the *Data Browser* or editing module code, are accessible via double click on the corresponding module in the graphical representation of the analysis. The analysis execution can also be launched from the *Analysis Designer*.

The *Analysis Designer* is based on the implementation of modular physics analysis described in the previous section. It visualizes the data flow between the modules of an analysis and provides access to any given module or parameter as sketched in Figure 3. Modules can be added to the analysis from a list of predefined or custom C++ and Python modules. The data flow is controlled by connecting sink and source ports of modules via drag and drop. Module parameters can be modified in a *Property View* that displays a table of parameter names and associated values. While the complete analysis handling and the design of the analysis data flow can be performed visually, the textual programming of the actual modules is performed in the user's favorite editor. For Python modules, the editor opens up with double clicking on a module. Analyses can be executed from the *Analysis Designer* via a button. The command line output of the analysis execution can then be monitored in a separate section of the window. The analysis execution happens in a separate process to keep the VISPA environment fully functional during execution.

The input and output data of an analysis can be explored within the same environment in the *Data Browser*. It visualizes all data in a PXL file, in particular decay trees of objects with relations, e.g., particles, and gives access to all contained information in a *Property View* as sketched in Figure 3. The views of the *Data Browser* represent data focused on its structure and full information coverage, as opposed to event displays which focus rather on the visualization in the context of detector geometries. The two main purposes of the *Data Browser* are the understanding of the input data and analysis debugging. By browsing the input data, all the available information can be

identified. By browsing the output data of intermediate or final analysis steps, the functionality of each module, which may filter the data or add information to it, can be checked.

Finally, dialogs to execute the analysis or to send it to a batch system are provided. For the submission of batch jobs, a dedicated plug-in is provided by VISPA, the *Batch Manager*. The *Batch Manager* plug-in allows to configure batch jobs where input parameters of analyses may be iterated. Hereby, one can execute slightly modified versions of a single analysis, a common use case in physics analysis. The jobs may be sent to either a plug-in for multicore application on a single machine (*Local Batch Manager*) or to a plug-in for submission to a Condor or Grid computing cluster (*Condor/Grid Batch Manager*), providing full flexibility for the execution of analyses from a single computer to large scale computing.

7. Example Applications

The applications of VISPA range from rather simple use cases to complex analyses. In this section we provide a few examples for such cases with different complexity which are performed with VISPA.

The `EasyROOTplot` [36] modules are a collection of four plotting modules and a text file parser which takes text files in Comma Separated Value (CSV) [34] format as input. They represent a plotting tool based on `ROOT` [13], which can be steered graphically in the *Analysis Designer*. An input module and plotting modules are plugged together, and parameters can be configured. The text file parser converts the values found in the CSV file into an N-dimensional vector that is used as input for the plotting modules. It allows the configuration via the *Property View* (Fig. 3) to adapt to differently formatted files such as different separators, or data values stored in rows instead of columns. The plotting modules then take these N-vectors to produce plots, such as graphs or histograms. The variables to be plotted can be chosen via the *Property View* and are identified by line headings in the CSV file or by identifier numbers generated at parsing time. The output can be chosen via a file save dialog and can be any picture format supported by `ROOT`. For further customization such as adapting the plotting style, the user can modify the corresponding Python code lines marked within the `EasyROOTplot` modules.

Another application of VISPA is browsing of Monte Carlo (MC) generator output in the form of the Les Houches Event (LHE) [33] data format. The LHE format contains a list of particles with mother-daughter relations representing a particle production/decay chain. In VISPA one can combine an LHE input module [37] and a PXL output module to convert the LHE objects to PXL objects. The resulting PXL file can be inspected with the *Data Browser*. In this way one can validate MC generator output which, due to its complexity with many related particles, can be best understood graphically and in an interactive manner. The zooming functionality of the graphical views in VISPA helps to understand large particle decay chains at both detailed and large-scale level.

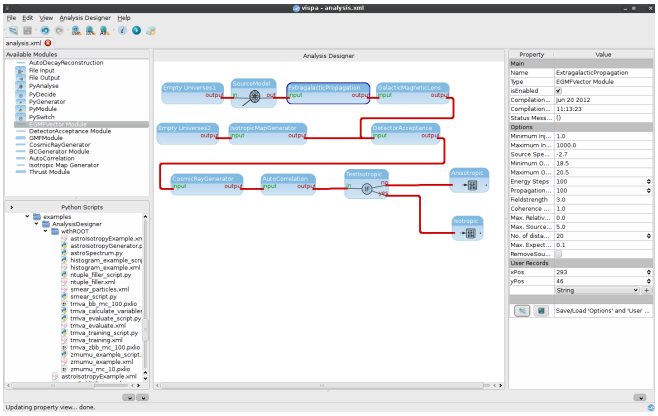
A plug-in related to the *Data Browser* is the *Event Editor*. This plug-in can be used, e.g., to graphically create a parton scattering process similar to a Feynman diagram. Particles can be selected from a list by drag and drop and can be connected to build mother-daughter relations between them. The diagram can be stored as an *Event* of the PXL physics library or exported to an image file. Such diagrams are used as templates ,e.g., for automated reconstruction of decay trees

from final state particles as implemented in a specialized module [25]. Typical applications at the LHC are the reconstruction of Z or W bosons, top quarks or hypothetical particles from extensions of the Standard Model of particle physics.

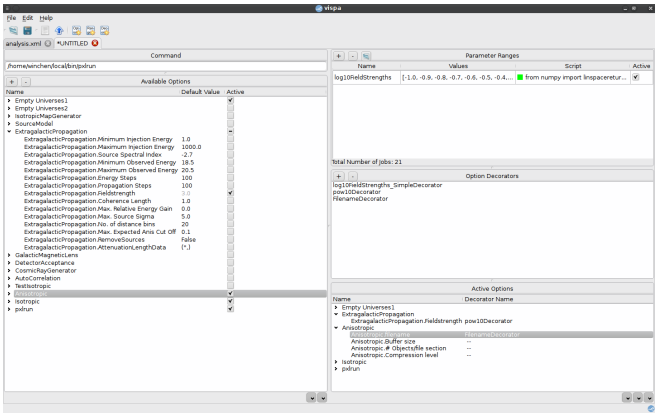
A more complex application of VISPA is the Parametrized Simulation Engine for Cosmic Rays (`PARSEC`) [38], a MC generator to model energy-dependent anisotropies in the UHECR arrival distribution. Simulated datasets are generated based on models of the source distribution and the propagation in extragalactic space as well as in our galaxy. The simulation code is separated into individual VISPA modules whose parameters can be accessed in the GUI. In a first step empty *BasicContainers* are generated and filled with *UHECRSources* according to a user defined model of the source distribution. Based on the source distribution, maps of the probability to observe a UHECR with an energy E from a direction (ϕ, θ) including deflection in extragalactic magnetic fields are calculated. The maps are stored as *BasicNVectors*. In the next step, these probability maps are transformed by multiplying the vectors with precalculated matrices to account for deflections in the coherent galactic magnetic field of the milky way. In the final step of the MC generator, individual cosmic rays are generated from these probability maps. The underlying models of the simulation can be modified by either changing the parameters accessible through the *Property View* or by changing the complete module. The used modules and parameters are stored in the containers as *UserRecords* to keep track of the settings. Besides these advantages of the modular design and GUI access, the *Data Browser* is frequently used to quickly check the simulation output in a typical run. Furthermore, the *Batch Manager* allows for the convenient mass production of UHECRs for large parameter scans. Figure 6 shows screenshots of a typical `PARSEC` analysis cycle with the *Analysis Designer*, the *Batch Manager* and the *Data Browser*.

Another rather complex use case in high energy physics is the application of multivariate data analysis techniques. Here, VISPA enables the dynamic integration of external packages designed for multivariate data analysis into one common analysis workflow. The PXL module system is used to interface the configuration of a corresponding package such as TMVA [39]. Sets of parameters used to configure a multivariate analysis package can be visualized and steered via the *Property View*, which allows to modify the most important and most used options. VISPA supports transparent and user-defined splitting of data streams into categories as used for training, testing and evaluation purposes. The data stream can be further divided into separate sub-samples of the available phase space, depending on the kinematics specific to particular physics processes. Categorizing data is particularly useful when several classifiers are needed, e.g., to build super-discriminants of diversified multivariate classifiers or to chain several classifiers. Technically, the classifier output can be conveniently stored as a *UserRecord* in each *Event*. Classifier output values using various configurations can be stored consecutively in the same *Event*. This feature allows to compare the performance of diversified classifiers and configurations. In this way, an optimization of the used classifiers with respect to consecutive analysis steps is feasible. An important example application is the selection of the classifier resulting in the best expected sensitivity of a particular analysis.

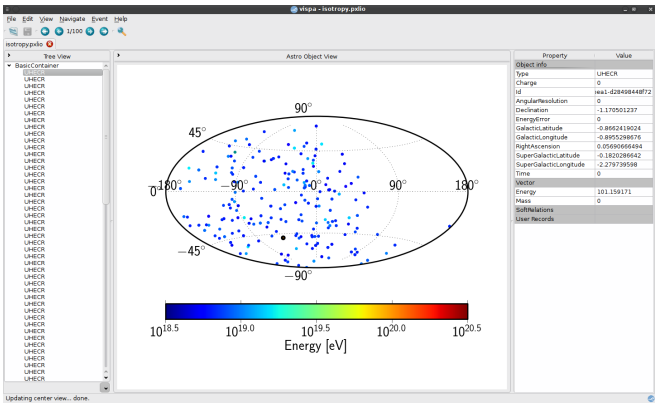
An example for an experiment-specific plug-in that integrates experiment-specific tasks of the analysis flow into the VISPA development environment is the `ConfigEditor` [40]. It is a widely used tool for inspecting the configuration of workflows in the CMS experiment and shares most of its GUI implementation with the *Analysis Designer*. For VISPA analyses the `ConfigEditor` is



(a)



(b)



(c)

Figure 6. (a) Exemplary simulation chain of the cosmic ray generator PARSEC in the *Analysis Designer*. Isotropically distributed cosmic rays and cosmic rays from point sources are generated and analyzed. The parameters of the extragalactic magnetic field model can be modified in the *Property View* on the right-hand side. (b) *Batch Job Designer* (part of the *Batch Manager* plug-in) to create multiple simulation jobs with varying strength of the extragalactic magnetic field. (c) Resulting arrival directions of isotropically distributed cosmic rays in galactic coordinates within a detector field of view, opened in the *Data Browser*.

used for the analysis step in which objects relevant to the analysis are configured and selected from a large variety of reconstructed objects available from the experiment. For CMS, it is generally useful for debugging of configurations of any step of the data processing chain of the experiment: trigger, reconstruction, event generation and simulation.

8. Conclusions

The VISPA graphical development environment provides the tools to iterate through a full analysis cycle. For designing and programming of physics analyses, three different software paradigms are combined. First, physics analyses are developed using visual programming. The development process combines visual design of the analysis flow in the graphical user interface and textual programming of individual modules. Second, in the context of flow-based programming, the data are handed through chains of modules that apply different algorithms implementing the analysis logic. Third, algorithms are designed with object-oriented textual programming based on the objects provided by PXL. VISPA also provides the tools for local or distributed execution, and for validation of the results.

To ensure a flexible analysis design and to cover all steps of physics analyses in different fields of physics, the VISPA user interface has been designed as a plug-in-based framework. This enables the extension of VISPA with more analysis-related tools and the application of VISPA in other fields of research besides HEP and astroparticle physics. Because of its platform-independent implementation and its well-defined analysis and data interfaces, VISPA particularly facilitates teamwork in analysis groups.

Acknowledgments

We wish to thank Benedikt Hegner for fruitful discussions and valuable comments on the manuscript. For important contributions to the first version of PXL, we thank Steffen Kappler. For fruitful discussions during the development phase of PXL, we thank Matthias Kirsch. We thank Carsten Hof, Philipp Biallass and Holger Pieta for careful evaluation of the PXL I/O system. We also thank Anna Henrichs for providing a PXL interface to the data format of the ATLAS experiment. For the creation of the user manual for PXL, we thank Oxana Actis. This work is supported by the Ministerium für Wissenschaft und Forschung, Nordrhein-Westfalen, the Bundesministerium für Bildung und Forschung (BMBF), and the Helmholz Alliance "Physics at the Terascale". T. Winchen gratefully acknowledges funding by the Friedrich-Ebert-Stiftung.

References

[1] M. Erdmann et al., *Visual Physics Analysis (VISPA)*, `http://vispa.physik.rwth-aachen.de`.

[2] M.Brodski et al., *Visual Physics Analysis - from desktop towards physics analysis at your fingertips*, *Proceedings ACAT2011, London, England, PoS(ACAT2011)* (2011).

[3] M.Brodski et al., *Visual Physics Analysis - Applications in High Energy and Astroparticle Physics*, *Proceedings ACAT2010, Jaipur, India, PoS(ACAT2010)* **064** (2010).

[4] M. Brodski et al., *Visual Physics Data Analysis in the Web Browser*, *Proc. Computing in High Energy Physics (CHEP2010), Taipei, Taiwan, J. Phys.: Conf. Ser.* **331** (2011) 072056.

[5] O. Actis et al., *Visual physics analysis VISPA*, *Proc. Computing in High Energy Physics (CHEP2009), Prag, Czech Republic, J. Phys.: Conf. Ser.* **219** (2010) 042041.

[6] O. Actis et al., *VISPA - Visual Physics Analysis on Linux, Mac OS X and Windows*, *Proceedings Europhysics Conference on High Energy Physics (EPS-HEP 2009), Krakow, Poland, PoS(EPS-HEP 2009)* **447** (2009).

[7] O.Actis et al., *Visual Physics Analysis (VISPA) - Concepts and First Applications*, *Proc. 34th Int. Conf. High Energy Physics (ICHEP 2008), Philadelphia, Pennsylvania* (2008) [arXiv:0810.3609].

[8] O. Actis et al., *VISPA: a Novel Concept for Visual Physics Analysis*, *Proceedings ACAT2008, Erice, Sicily, PoS(ACAT08)* (2008) 070.

[9] S. Kappler et al., *Concepts, developments and advanced applications of the PAX toolkit*, (2006) [physics/0605063].

[10] C. D. Jones et al., *The New CMS Event Data Model and Framework*, *Proc. Computing in High Energy Physics (CHEP2006), Mumbai, India* (2006).

[11] P. Calafiura, W. Lavrijsen, C. Leggett, M. Marino, and D. Quarrie, *The athena control framework in production, new developments and lessons learned*, *Proc. Computing in High-Energy Physics (CHEP '04), Interlaken, Switzerland* (2005) https://cdsweb.cern.ch/record/865624.

[12] S. Argiro et al., *The Offline Software Framework of the Pierre Auger Observatory*, *Proc. IEEE Nuclear Science Symposium, Medical Imaging Conference, Puerto Rico* (2005).

[13] R. Brun and F. Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, *Nucl. Inst. & Meth. in Phys. Res. A* **398** (1996) 81-86 http://root.cern.ch.

[14] M. Erdmann et al., *Physics eXtension Library (PXL)*, https://forge.physik.rwth-aachen.de/projects/pxl/wiki/.

[15] Riverbank Computing Limited, *PyQt*, http://www.riverbankcomputing.co.uk.

[16] Nokia Corporation, *Qt - Cross-platform application and UI framework*, http://qt.nokia.com/.

[17] M. Erdmann et al., *VISPA Class Documentation*, https://forge.physik.rwth-aachen.de/public/vispa/0.6/.

[18] S. Kappler et al., *The PAX Toolkit and its Applications at Tevatron and LHC*, *IEEE Trans. Nucl. Sci.* **53** (2006) 506, [physics/0512232].

[19] A. Schmidt et al., *New applications of PAX in physics analyses at hadron colliders*, *Proc. Computing in High Energy Physics (CHEP2004), Interlaken, Switzerland, CERN-2005-002* (2005) 317.

[20] M. Erdmann et al., *Physics analysis expert PAX: First applications*, *Proc. Computing in High Energy Physics (CHEP2003), La Jolla, California* (2003) [physics/0306085].

[21] M. Erdmann et al., *User oriented design in high energy physics applications: Physics analysis expert*, in *Proceedings of the 14th Topical Conference on Hadron Collider Physics (HCP 2002)*, pp. 494-497, 2002.

[22] M. Erdmann et al., *PXL Class Documentation*, https://forge.physik.rwth-aachen.de/public/pxl/3.2/cplusplus/.

[23] P. Leach, M. Mealling, and R. Salz, *A Universally Unique IDentifier (UUID) URN Namespace*, *Request for Comments* **4122** (2005).

[24] D. Beazley, *Simplified Wrapper and Interface Generator (SWIG)*, `http://www.swig.org`.

[25] O. Actis et al., *Automated Reconstruction of Particle Cascades in High Energy Physics Experiments*, `arXiv:0801.1302`.

[26] J.-P. Lang, *Redmine*, `http://www.redmine.org`.

[27] M. Mackall, *Mercurial*, `http://mercurial.selenic.com`.

[28] E. Jones, T. Oliphant, and P. Peterson, *SciPy: Open Source Scientific Tools for Python*, (2001) `http://www.scipy.org`.

[29] J. Hunter, *matplotlib*, `http://matplotlib.sourceforge.net`.

[30] K. Cranmer et al., *Roostats Manual*, `http://root.cern.ch/root/html/ROOFIT_ROOSTATS_Index.html`.

[31] D. Kirkby and W. Verkerke, *Roofit Manual*, `http://root.cern.ch/drupal/content/roofit`.

[32] T. Oliphant, *Python for Scientific Computing*, *Computing in Science and Engineering* **9** (2007), no. 3 10-20, `http://numpy.scipy.org`.

[33] J. Alwall et al., *A standard format for Les Houches Event Files*, *Computer Physics Communications* **176** (2007), no. 4 300-304.

[34] Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, *Request for Comments* **4180** (2005).

[35] M. Erdmann and P. Schiffer, *Measuring Cosmic Magnetic Fields with Ultra High Energy Cosmic Ray Data*, *Astropart. Phys.* **33** (2010) 201, [`arXiv:0904.4888`].

[36] M. Erdmann et al. `https://forge.physik.rwth-aachen.de/projects/easyrootplot`.

[37] M. Erdmann et al. `https://forge.physik.rwth-aachen.de/projects/pxl-lhe-input/wiki`.

[38] H.-P. Bretz, M. Erdmann, P. Schiffer, and T. Winchen, *PARametrized Simulation Engine for Cosmic rays (PARSEC)*, `http://www.physik.rwth-aachen.de/parsec`.

[39] H. Voss et al., *TMVA, the Toolkit for Multivariate Data Analysis with ROOT*, *Proceedings ACAT2007, Amsterdam, Netherland, PoS(ACAT2007)* **040** (2007).

[40] A. Hinzmann, *Visualization of the CMS Python Configuration System*, *Proc. 17th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2009), Prague, Czech Republic, J. Phys.: Conf. Ser.* **219** (2009) 042008 `http://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookConfigEditor`.